DOUBLE-EXPONENTIATION IN FACTOR-4 GROUPS AND ITS APPLICATIONS

KORAY KARABINA

ABSTRACT. In previous work we showed how to compress certain prime-order subgroups of the cyclotomic subgroups of orders $2^{2m}+1$ of the multiplicative groups of $\mathbb{F}_{2^{4m}}^*$ by a factor of 4. We also showed that single-exponentiation can be efficiently performed using compressed representations. In this paper we show that double-exponentiation can be efficiently performed using factor-4 compressed representation of elements. In addition to giving a considerable speed up to the previously known fastest single-exponentiation algorithm for general bases, double-exponentiation can be used to adapt our compression technique to ElGamal type signature schemes.

1. INTRODUCTION

The Diffie-Hellman key agreement protocol [3] can be used by two parties A and B to establish a shared secret by communicating over an unsecured channel. Let $G=\langle g\rangle$ be a prime-order subgroup of the multiplicative group $\mathbb{F}_q^*$ of a finite field $\mathbb{F}_q$. Party A selects a private key a and sends g^a to B. Similarly, B selects a private key b and sends g^b to A. Both parties can then compute the shared secret g^{ab}. Security of the protocol depends on the intractability of the problem of computing a from g^a; this is called the discrete logarithm problem in G. If q is prime (say $q=p$), then the fastest algorithms known for solving the discrete logarithm problem in G are Pollard's rho method [17] and the number field sieve [9]. To achieve a 128-bit security level against these attacks, one needs to select $\#G\approx 2^{256}$ and $p\approx 2^{3072}$ [6, Section 4.2]. Note that even though the order of G is approximately 2^{256}, the natural representation of elements of G, namely as integers modulo p, are approximately 3072 bits in length. If q is a power of 2 or 3, then one needs to select $\#G\approx 2^{256}$ and $q\approx 2^{4800}$ to achieve 128-bit security level against Pollard's rho method and Coppersmith's index-calculus attack [2, 12]. This brings an overhead both to the efficiency of the protocol and to the number of bits that need to be stored or transmitted.

In recent years, there have been several proposals for compressing the elements of certain subgroups of certain finite fields [21, 8, 1, 13, 18, 5, 4, 20, 11]. The compression methods in these works fall into two categories. They either use a rational parametrization of an algebraic torus [18, 5, 4], or use the trace representation of elements [21, 8, 1, 13, 20, 11]. Even though there is a close relation between these two methods (see [18]), the trace representation of elements seems to give more efficient single-exponentiation algorithms. Single-exponentiation in the context of compressed representations is the operation of computing the compressed representation of g^a given an integer a and the compressed representation of g. Double-exponentiation in the context of compressed representations is the operation of

Date: September 25, 2009.
Key words and phrases. Finite field compression, cyclotomic subgroups, double-exponentiation.

computing the compressed representation of g^{ak+bl} given integers a, b, and the compressed representations of g^k, g^l. It is not clear how to perform double-exponentiation *efficiently* when one uses the trace representation of elements because the trace function is not multiplicative. On the other hand, the use of a rational parametrization of a torus enjoys the full functionality of the group structure, and one can perform double-exponentiation with similar efficiency to that of a single-exponentiation.

Having efficient double-exponentiation is crucial in cryptographic applications. For example, in ElGamal type signature schemes the verifier should perform a double-exponentiation to verify the signature on the received message. Moreover, double-exponentiation can be used to speed up single-exponentiation by representing the exponent $\tau = ak + b$ where k is some fixed integer and a, b are half the bitlength of τ. Then g^k can be precomputed and given τ, one can compute $g^\tau = (g^k)^a \cdot g^b$ using simultaneous exponentiation (*Straus-Shamir's trick*; see Algorithm 14.88 in [14]) much more efficiently than direct exponentiation by τ. As mentioned in the previous paragraph, it is not clear if one can favourably exploit this idea when the trace representation of elements is used.

Lenstra and Stam [22] show that in the case of factor-2 and factor-3 compression in large-prime characteristic fields one can perform double-exponentiation very efficiently and they discuss some related applications such as speeding up the single-exponentiation algorithm for compressed elements.

In this paper, we show that double-exponentiation that works directly with factor-4 compressed elements can be performed very efficiently in the case of characteristic two fields, and describe two particular cryptographic applications. As a first application, we show how to use double-exponentiation to speed up single-exponentiation thereby obtaining an estimated 20% acceleration over the previously known fastest single-exponentiation algorithm when the base is general. Speeding up the single-exponentiation is important as it speeds up some cryptographic protocols using the factor-4 compression technique. For example, as observed in [11], the factor-4 compression technique can be applied to the image of the symmetric bilinear pairing derived from an embedding degree $k = 4$ supersingular elliptic curve defined over a characteristic two field. If this pairing is used to implement the identity-based key aggrement protocol of Scott [19], then the messages exchanged can be compressed by a factor of 4; moreover, the single-exponentiation in the protocol can be performed using the compressed representation of elements. As a second application, we give details on deploying factor-4 compressed representation of elements in the Nyberg-Rueppel signature scheme [16]; our method also reduces the size of public keys.

The remainder of the paper is organized as follows. Section 2 introduces some terminology and sets the notation that we will use throughout the paper. In Section 3 we review the previous work on factor-4 compression. Section 4 presents our double-exponentiation algorithm and an analysis of the algorithm. Two cryptographic applications of our double-exponentiation algorithm are given in Section 5. We make some concluding remarks in Section 6.

2. Preliminaries and Notation

Let q be a prime power and $\mathbb{F}_q$ denote a finite field with q elements. Let n be a prime such that $\gcd(n, q) = 1$, and let k be the smallest positive integer such that $q^k \equiv 1 \pmod{n}$. Then $\mathbb{F}_{q^k}^*$ has a multiplicative subgroup of order n which cannot be embedded in the multiplicative

group of any extension field $\mathbb{F}_{q^i}$ for $1 \leq i < k$. For such a triple (q, k, n) we denote the multiplicative group of order n by μ_n and call k the *embedding degree* of μ_n over $\mathbb{F}_q$. In this setting, we let h be a positive integer and define $t_h = q + 1 - h \cdot n$ to be the *trace* of μ_n over $\mathbb{F}_q$ with respect to the *cofactor* h. Throughout the rest of this paper we will assume that the cofactor h is fixed and we simply denote the trace of μ_n by t, instead of t_h.

Let $g \in \mathbb{F}_{q^k}$ and let s be a positive divisor of k. We assume that g is not contained in any proper subfield of $\mathbb{F}_{q^k}$. The *conjugates* of g over $\mathbb{F}_{q^s}$ are $g_i = g^{q^{is}}$ for $0 \leq i < k/s$. The *trace* of g over $\mathbb{F}_{q^s}$ is the sum of the conjugates of g over $\mathbb{F}_{q^s}$, i.e.,

$$\mathrm{Tr}_s(g) = \sum_{i=0}^{\frac{k}{s}-1} g_i \in \mathbb{F}_{q^s}. \tag{2.1}$$

The *minimal polynomial* of g over $\mathbb{F}_{q^s}$ is the monic polynomial

$$f_{g,s}(x) = \prod_{i=0}^{\frac{k}{s}-1} (x - g_i). \tag{2.2}$$

Note that $f_{g,s}(x) \in \mathbb{F}_{q^s}[x]$. When $s = 1$ we simply use $\mathrm{Tr}(g)$ and $f_g(x)$ by abuse of notation. Also, we will assume that the conjugates of g over $\mathbb{F}_{q^s}$ are well defined for any integer i by setting $g_i = g_{i \bmod k/s}$.

We fix some notation for finite field operations that will be used in the remainder of the paper. We will denote by $A_i, a_i, C_i, F_i, I_i, M_i, m_i$, and S_i the operations of addition, addition by 1 or 2, cubing, exponentiation by a power of the characteristic of the field, inversion, multiplication, multiplication by 2, and squaring in $\mathbb{F}_{q^i}$ for $i \in \{1, 2\}$. $SR_{i,j}$ will denote the cost of finding a root of a degree i irreducible polynomial over $\mathbb{F}_{q^j}$. We use *soft-O* notation $\tilde{O}(\cdot)$ as follows: $a = \tilde{O}(b)$ if and only if $a = O(b(\log_2 b)^c)$ for some constant c.

3. Review of Factor-4 Compression

We recall some of the facts on factor-4 compression and also review the exponentiation algorithms presented in [11].

Let r be a positive integer, and let $q = 2^{2r+1}$, $t = \pm 2^{r+1}$, $T = |t|$. The values of r for which $q + 1 - t = hn$ and n is prime lead to a multiplicative subgroup μ_n of $\mathbb{F}_{q^4}^*$ of prime order n with embedding degree 4 and trace t. We fix h, n, q, t, T and $\mu_n = \langle g \rangle$ in this fashion, and also write $c_u = \mathrm{Tr}(g^u)$. Note that $c_0 = 0$ and $c_{uT} = c_u^T$.

The following theorem shows that $g^u \in \mu_n$ can be uniquely represented (up to conjugation over $\mathbb{F}_q$) by its trace, thus providing compression by a factor 4. From now on, we will refer to this group $\mu_n \subset \mathbb{F}_{q^4}^*$ as the *factor-4 subgroup* of $\mathbb{F}_{q^4}^*$ with trace t.

Theorem 3.1. [11, Corollary 4.5] *Let $\mu_n = \langle g \rangle$ be the factor-4 subgroup of $\mathbb{F}_{q^4}^*$ with trace t. Let $f_{g^u}(x)$ be the minimal polynomial of $g^u \in \mu_n$ over $\mathbb{F}_q$. Then*

$$f_{g^u}(x) = x^4 + c_u x^3 + c_u^T x^2 + c_u x + 1.$$

Next, we recall some facts from [11] that we will use in Section 4.

Lemma 3.2. *Let $\mu_n = \langle g \rangle$ be the factor-4 subgroup of $\mathbb{F}_{q^4}^*$ with trace t. Then for all integers u and v we have*
(i) $c_{u+v} = (c_u + c_{u-2v})c_v + c_{u-v}c_v^T + c_{u-3v}$ [11, Corollary 4.4(i)].
(ii) $c_u = c_{-u}$ [11, Lemma 4.2(i)].
(iii) $c_{2u} = c_u^2$ [11, Corollary 4.4(ii)].
(iv) $c_u c_v = c_{u+v} + c_{u-v} + c_{u+v(t-1)} + c_{v+u(t-1)}$ [11, Lemma 4.2(ii)].

Remark 3.3. Throughout the remainder of this paper, we will assume without loss of generality that the trace t is positive. If t is negative then one can replace the expressions of the form $c_u^{p(t)}$, where p is some polynomial, by $c_u^{p(T)}$ without changing the validity of the results in this paper.

Let a, b be integers with $0 < a, b < n$. Single-exponentiation to the base a in μ_n is the operation of computing c_{ab} given c_a and b. Five single-exponentiation algorithms were presented in [11]. We concentrate on Algorithm 1 that works directly with c_a, and is the fastest exponentiation algorithm for general bases. For completeness we present Algorithm 1 and its running time.

Algorithm 1 Single-exponentiation
Input: c_a and b
Output: c_{ab}

Write $b = \sum_{i=0}^{\ell-1} b_i 2^i$ where $b_i \in \{0,1\}$ and $b_{\ell-1} = 1$
$s_u = [c_{u-2}, c_{u-1}, c_u, c_{u+1}] \leftarrow [c_a, 0, c_a, c_a^2]$
$m_1 \leftarrow 1/c_a^{t+1}$ and $m_2 \leftarrow 1/c_a$
for i **from** $\ell - 2$ **down to** 0 **do**
  $c_{2u-1} \leftarrow m_1 \left((c_{u+1} + c_u + c_{u-1} + c_{u-2})^2 + (c_u + c_{u-1})^2(c_a^t + c_a^2)\right)$
  $c_{2u} \leftarrow c_u^2$
  $c_{2u+1} \leftarrow c_{2u-1} + m_2 \left((c_{u+1} + c_{u-1})^2 + c_u^2 c_a^t\right)$
  if $b_i = 1$ **then**
    $c_{2u+2} \leftarrow c_{u+1}^2$
    $s_u \leftarrow [c_{2u-1}, c_{2u}, c_{2u+1}, c_{2u+2}]$
  else
    $c_{2u-2} \leftarrow c_{u-1}^2$
    $s_u \leftarrow [c_{2u-2}, c_{2u-1}, c_{2u}, c_{2u+1}]$
  end if
end for
Return (c_u)

TABLE 3.1. Cost of Algorithm 1 for factor-4 compression. The exponent is an ℓ-bit integer.

Algorithm	Precomputation	Main Loop
Algorithm 1 [11]	$1I_1 + 1M_1$	$(4M_1 + 4S_1)(\ell - 1)$

Next, we give a generalization of Theorem 4.7 in [11] that will be used in Section 4 to describe a double-exponentiation algorithm in μ_n.

Theorem 3.4. *Let $\mu_n = \langle g \rangle$ be the factor-4 subgroup of $\mathbb{F}_{q^4}^*$ with trace t. Let $c_u = \mathrm{Tr}(g^u)$,*

$$A = \begin{pmatrix} c_v & c_v & 0 & 0 \\ 0 & c_v & c_v & 0 \\ 0 & 1 & c_v^t & 1 \\ 1 & c_v^t & 1 & 0 \end{pmatrix},\ X = \begin{pmatrix} c_{2u-3v} \\ c_{2u-v} \\ c_{2u+v} \\ c_{2u+3v} \end{pmatrix},\ Y = \begin{pmatrix} (c_u + c_{u-2v})^2 + c_{u-v}^2 c_v^t \\ (c_{u+v} + c_{u-v})^2 + c_u^2 c_v^t \\ (c_u + c_{u+v})^2 c_v \\ (c_{u-v} + c_u)^2 c_v \end{pmatrix}.$$

Then
(i) A is invertible and $AX = Y$.
(ii) If c_v is given then A and A^{-1} can be efficiently computed.
(iii) $c_{2u-v} = \frac{1}{c_v^{t+1}} \left((c_{u+v} + c_u + c_{u-v} + c_{u-2v})^2 + (c_u + c_{u-v})^2 (c_v^t + c_v^2) \right)$.

Proof. The proof is analogous to the proof of Theorem 4.7 in [11]. □

Corollary 3.5. *Let $\mu_n = \langle g \rangle$ be the factor-4 subgroup of $\mathbb{F}_{q^4}^*$ with trace t. Let c_v, c_{2u-v} and $s_{u,v} = [c_{u-2v}, c_{u-v}, c_u, c_{u+v}]$ be given. Then*
(i) $c_{u+2v} = (c_{u+v} + c_{u-v})c_v + c_u c_v^t + c_{u-2v}$, and can be computed at a cost of $1F_1 + 2M_1$.
(ii) $c_{2u+v} = (c_{u+v} + c_{u-v})c_u + c_v c_u^t + c_{2u-v}$, and can be computed at a cost of $1F_1 + 2M_1$.
(iii) $c_{u-3v} = (c_u + c_{u-2v})c_v + c_{u-v} c_v^t + c_{u+v}$, and can be computed at a cost of $1F_1 + 2M_1$.
(iv) $c_{3u-v} = (c_v + c_{2u-v})c_u + c_{u-v} c_u^t + c_{u+v}$, and can be computed at a cost of $1F_1 + 2M_1$.
(v) $c_{u-4v} = c_{u+2v} + (c_u + c_{u-2v})(c_v^2 + c_v^t + 1) + c_{u-v} c_v^{t+1}$, and (c_{u+2v}, c_{u-4v}) can be computed at a cost of $1F_1 + 5M_1 + 1S_1$.
(vi) $c_{4u-v} = c_{2u+v} + (c_v + c_{2u-v})(c_u^2 + c_u^t + 1) + c_{u-v} c_u^{t+1}$, and (c_{2u+v}, c_{4u-v}) can be computed at a cost of $1F_1 + 5M_1 + 1S_1$.
(vii) $c_{3u+v} = c_{3u-v} + (c_{u+v} + c_{u-v})(c_u^2 + c_u^t + 1) + c_v c_u^{t+1}$, and (c_{3u-v}, c_{3u+v}) can be computed at a cost of $1F_1 + 5M_1 + 1S_1$.

Proof. (i)–(iv) follow from Lemma 3.2(i) and (ii).
(v) Using Lemma 3.2(i) and (ii), we first write

$$c_{u+2v} = (c_{u+v} + c_{u-v})c_v + c_u c_v^t + c_{u-2v} \tag{3.1}$$
$$c_{u-4v} = (c_{u-v} + c_{u-3v})c_v + c_{u-2v} c_v^t + c_u. \tag{3.2}$$

Now, adding (3.1) and (3.2) together and replacing $c_{u-3v} = (c_u + c_{u-2v})c_v + c_{u-v} c_v^t + c_{u+v}$ gives $c_{u-4v} = c_{u+2v} + (c_u + c_{u-2v})(c_v^2 + c_v^t + 1) + c_{u-v} c_v^{t+1}$. Finally, once c_{u+2v} is computed as in (i) at a cost of $1F_1 + 2M_1$, it is clear that c_{u-4v} can be computed at a cost of $3M_1 + 1S_1$, which completes the proof.
(vi) The proof follows as in (v) after interchanging u and v.
(vii) The proof follows as in (v) after replacing (u, v) by $(u - v, u)$. □

4. Double-Exponentiation in Factor-4 Groups

Definition 4.1. Let $\mu_n = \langle g \rangle$ be the factor-4 subgroup of $\mathbb{F}_{q^4}^*$ with trace t, and as usual let $c_u = \mathrm{Tr}(g^u)$. Let a, b, k, l be integers with $0 < a, b, k, l < n$. *Double-exponentiation* to the base (k, l) in μ_n is the operation of computing c_{ak+bl} given a, b, c_l and $s_{k,l} = [c_{k-2l}, c_{k-l}, c_k, c_{k+l}]$.

We assume that a, b, k, l are strictly positive as otherwise double-exponentiation just becomes a single-exponentiation. Note that k and l are not necessarily known.

TABLE 4.1. Update rules for double-exponentiation in factor-4 groups

Rule	Condition	d	e	u	v	c_v	c_{2u-v}	$s_{u,v} = [c_{u-2v}, c_{u-v}, c_u, c_{u+v}]$
if $d \geq e$								
R1	$d \leq 4e$	$d-e$	e	u	$u+v$	c_{u+v}	c_{u-v}	$[c_{u+2v}, c_v, c_u, c_{2u+v}]$
R2	$d \equiv e \pmod 2$	$(d-e)/2$	e	$2u$	$u+v$	c_{u+v}	c_{3u-v}	$[c_v^2, c_{u-v}, c_u^2, c_{3u+v}]$
R3	$d \equiv 0 \pmod 2$	$d/2$	e	$2u$	v	c_v	c_{4u-v}	$[c_{u-v}^2, c_{2u-v}, c_u^2, c_{2u+v}]$
R4	$e \equiv 0 \pmod 2$	d	$e/2$	u	$2v$	c_v^2	c_{u-v}^2	$[c_{u-4v}, c_{u-2v}, c_u, c_{u+2v}]$
else								
S	$e > d$	e	d	v	u	c_u	c_{u-2v}	$[c_{2u-v}, c_{u-v}, c_v, c_{u+v}]$

TABLE 4.2. Analysis of update rules for double-exponentiation in factor-4 groups

Rule	Condition	Cost	Reduction factor for $(d+e)$
if $d \geq e$			
R1	$d \leq 4e$	$2F_1 + 4M_1$	$\geq 5/4,\ < 2$
R2	$d \equiv e \pmod 2$	$1F_1 + 5M_1 + 2S_1$	2
R3	$d \equiv 0 \pmod 2$	$1F_1 + 5M_1 + 2S_1$	$\geq 5/3,\ < 2$
R4	$e \equiv 0 \pmod 2$	$1F_1 + 5M_1 + 2S_1$	$> 1,\ < 10/9$
else			
S	$e > d$	0	1

A double-exponentiation algorithm was presented by Stam and Lenstra [22] for second-degree and third-degree recursive sequences. Their algorithm is an adaptation of Montgomery's method [15] to compute second-degree recursive sequences. In this section, we adapt the techniques used in [22, 15] and present a double-exponentiation algorithm for fourth-degree recursive sequences.

Algorithm 2 starts with $u = k, v = l, d = a > 0$, $e = b > 0$, from which it follows that $ud + ve = ak + bl = \tau$, c_v and $s_{u,v}$ are known, and c_{2u-v} can be computed at a cost of $1F_1 + 1I_1 + 3M_1 + 3S_1$ by Theorem 3.4. The reason we introduce the term c_{2u-v} in step 2 of Algorithm 2 is to avoid the repeated computation of c_{2u-v} during the updates. In the main part of the algorithm, u, v, d, e will be updated so that $ud + ve = ak + bl = \tau$ holds, $d, e > 0$, and $(d + e)$ decreases until $d = e$. Also, c_v, c_{2u-v} and $s_{u,v}$ are updated according to the new values of u and v. When $d = e$, we will have $\tau = d(u + v)$ and $s_{u,v} = [c_{u-2v}, c_{u-v}, c_u, c_{u+v}]$. Finally, we compute $c_\tau = c_{d(u+v)}$ using a single-exponentiation algorithm. Table 4.1 gives the update rules for $u, v, d, e, c_v, c_{2u-v}$ and $s_{u,v}$. Table 4.2 gives the cost of each update operation and the factor by which each update reduces $(d+e)$. The cost analysis as summarized in the third column of Table 4.2 follows from Table 4.1 and Corollary 3.5.

Correctness: Let $m = \gcd(a, b)$, $a = ma'$, and $b = mb'$. It is easy to see that if $m = 2^r m'$, $r \geq 0$, and m' is odd then after step 6 in Algorithm 2 we will have $f = 2^r$, $d = m'a'$, $e = m'b'$. Moreover, after step 9, we will have $d = e = m'$ and $d(u + v) = m'a'k + m'b'l$ since $ud + ve$ is kept invariant while applying the rules in Table 4.1. Hence, $ak + bl = 2^r m'a'k + 2^r m'b'l = f \cdot d(u + v)$, $c_{ak+bl} = c_{f \cdot d(u+v)}$, and the correctness of the algorithm follows from Lemma 3.2(iii), as $c_{f \cdot d(u+v)} = c^f_{d(u+v)}$.

Algorithm 2 Double-exponentiation
Input: $a > 0, b > 0, c_l$ and $s_{k,l} = [c_{k-2l}, c_{k-l}, c_k, c_{k+l}]$
Output: c_{ak+bl}

1: $u \leftarrow k,\ v \leftarrow l,\ d \leftarrow a,\ e \leftarrow b,\ s_{u,v} \leftarrow s_{k,l}$
2: $c_{2u-v} \leftarrow \frac{1}{c_v^{t+1}}\left((c_{u+v} + c_u + c_{u-v} + c_{u-2v})^2 + (c_u + c_{u-v})^2(c_v^t + c_v^2)\right)$
3: $f \leftarrow 1$
4: **while** d and e are both even **do**
5: $\quad d \leftarrow d/2,\ e \leftarrow e/2,\ f \leftarrow 2f$
6: **end while**
7: **while** $d \neq e$ **do**
8: $\quad$ Execute the first applicable rule in Table 4.1
9: **end while**
10: Compute $c_{d(u+v)}$ using Algorithm 1 with input c_{u+v} and d
11: Return $c_{d(u+v)}^f$

Analysis: The running time analysis of Algorithm 2 is similar to that in [15]. We will ignore the costs F_1 and S_1. If R4 is never required during the execution of Algorithm 2, then it is clear from Table 4.2 that the cost of the second while loop in Algorithm 2 never exceeds

$$\max(4\log_{5/4}(a'+b'), 5\log_2(a'+b'), 5\log_{5/3}(a'+b'))M_1 \approx 12.4\log_2(a'+b')M_1.$$

Now, suppose that R4 is used $i > 0$ times successively, with the starting (d, e) value (d_1, e_1). That is, $d_1 > 4e_1$, $d_1 \equiv 1 \pmod 2$, and $e_1 \equiv 0 \pmod 2$. Let (d_2, e_2) be the updated value of (d, e) after i applications of R4. Clearly, $d_1 = d_2$ and $e_1 = e_2 2^i$. Now, only the rule R2 is applicable, and suppose we apply R2 and R3 (possibly after R2) j times until R1 qualifies (i.e. $d \leq 4e$) or $j \leq i$; and suppose that the (d, e) value is updated to (d_3, e_3). Clearly, $e_2 = e_3$ and $d_3 \leq d_2/2^i$. If $d_3 \leq 4e_3$ then

$$\frac{(d_1 + e_1)}{(d_3 + e_3)} \geq \frac{5e_1}{5e_3} = 2^i.$$

If $j = i$ then

$$\frac{(d_1 + e_1)}{(d_3 + e_3)} \geq \frac{d_2 + e_2 2^i}{d_2/2^j + e_2} = 2^i.$$

In both cases, the value $(d + e)$ value is reduced at least by a factor of 2^i at a cost of $5(i + j)M_1 \leq 10iM_1$. Hence, the total cost of the second while loop in Algorithm 2 never exceeds $12.4\log_2(a' + b')M_1$. Using Algorithm 1 for step 10 in Algorithm 2, we can conclude that the total cost of Algorithm 2 never exceeds $12.4\log_2(a + b)M_1$.

In our experiments we observed that the performance of Algorithm 2 in practice is remarkably better than the upper bound $12.4\log_2(a + b)M_1$. Moreover, the behavior of Algorithm 2 becomes very stable as $(a+b)$ gets larger. We tested the performance of Algorithm 2 with 2^{20} randomly chosen pairs (a, b) such that $a \in [1, 2^{609}), b \in [1, 2^{612})$ and $a, b \in [1, 2^{1221})$. The intervals $([1, 2^{609}), [1, 2^{612}))$ are relevant for obtaining a faster single-exponentiation algorithm at the 128-bit security level (see Section 5.1), and the interval $[1, 2^{1221})$ is relevant for deploying our double-exponentiation algorithm in the Nyberg-Rueppel signature scheme at the 128-bit security level (see Section 5.2). Our experimental evidence suggests that the main loop in steps 7–9 of Algorithm 2 is executed approximately $1.45\log_2(A + B)$ times on average (where $A = a/\text{gcd(a,b)}$, $B = b/\text{gcd(a,b)}$), and that the average number of multiplications per iteration is 4.39. Moreover, R1 is used in around 61% of the total number

of iterations. In our experiments, as one might expect, $\gcd(a,b)$ is very small and the cost of step 10 is $2.32M_1$ on average (see Table 4.3). Note that step 11 can be performed at a negligible cost. Hence, we may conjecture that the expected running time of Algorithm 2 is $(1.45 \cdot (0.61 \cdot 4 + 0.39 \cdot 5)) \log_2(a+b)M_1 \approx 6.37 \log_2(a+b)M_1$.

TABLE 4.3. Practical behavior of Algorithm 2 at the 128-bit security level. 2^{20} pairs (a,b) were randomly chosen such that $a \in [1, 2^{609}), b \in [1, 2^{612})$ and $a, b \in [1, 2^{1221})$. $A = a/\gcd(a,b)$ and $B = b/\gcd(a,b)$.

	Average		Standard deviation	
	$a \in [1,2^{609})$, $b \in [1,2^{612})$	$(a,b) \in [1,2^{1221})$	$a \in [1,2^{609})$, $b \in [1,2^{612})$	$(a,b) \in [1,2^{1221})$
$\log_2(a+b)$	611.4	1221	1.058	0.816
$\log_2(A+B)$	611	1221	1.251	1.053
# of iterations (of steps 7–9)	$1.451 \cdot \log_2(A+B)$	$1.453 \cdot \log_2(a+b)$	0.018	0.012
# of multiplications per iteration (during steps 7–9)	4.390	4.390	0.027	0.019
# of multiplications in step 10	2.320	2.315	5.414	5.412
Rule	Average usage		Standard deviation	
R1	0.610	0.610	0.027	0.019
R2	0.175	0.175	0.013	0.009
R3	0.129	0.129	0.012	0.008
R4	0.085	0.085	0.013	0.009

5. Applications of Double-Exponentiation in Factor-4 Groups

In this section we discuss two applications of double-exponentiation of compressed elements in factor-4 groups. We show in Section 5.1 that double-exponentiation can be used to obtain faster single-exponentiation of compressed elements in factor-4 groups. Next, we show in Section 5.2 that double-exponentiation allows us to use compression techniques in ElGamal type signature schemes and furthermore obtain shorter public keys. As an illustrative example, we provide details for the Nyberg-Rueppel signature scheme.

Throughout this section we let $\mu_n = \langle g \rangle$ be the factor-4 subgroup of $\mathbb{F}_{q^4}^*$ with trace t, and let $T = |t|$.

5.1. **Speeding up Single-Exponentiation.** Algorithm 2 can be turned into a single-exponentiation algorithm as follows. Suppose we want to compute c_{rs} given c_r and $0 \le s < n$. Clearly, if $s = 0$ then $c_{rs} = 0$. Suppose $s \neq 0$, and write $s = aT + b$ where $0 \le a \le T$ and $0 \le b < T$ (this can be done as $T = \sqrt{2q}$ and $n = q + 1 - t$). If $a = 0$ then $c_{rs} = c_{rb}$ can be computed using Algorithm 1. If $b = 0$ then $c_{rs} = c_{raT} = c_{ra}^T$ (recall that T is a power of 2) can be computed by first computing c_{ra} from c_r and a using Algorithm 1, and then raising the result to the T'th power. Now, suppose $a, b \neq 0$. Then $c_{rs} = c_{arT+br} = c_{ak+bl}$, where $k = rT$ and $l = r$. In other words, c_{rs} can be computed from $a, b, c_l = c_r$ and $s_{k,l} = [c_{r(T-2)}, c_{r(T-1)}, c_{rT}, c_{r(T+1)}]$ using Algorithm 2. Note that c_l is already given, and $s_{k,l}$ is the last s_u value obtained from running Algorithm 1 with input c_r and T. Our discussion yields Algorithm 3 for single exponentiation in μ_n.

Let $C_1(i)$ denote the cost of Algorithm 1 when the input exponent b is approximately an $\lfloor i \rfloor$-bit integer. Similarly, let $C_2(i)$ denote the cost of Algorithm 2 when the sum of

Algorithm 3 Single-exponentiation
Input: c_r and s
Output: c_{rs}

1: **if** $s = 0$ **then**
2: $c_{rs} \leftarrow 0$
3: **else**
4: Write $s = aT + b$, where $T = |t|$, $0 \leq a \leq T$, $0 \leq b < T$
5: **if** $a = 0$ **then**
6: Use Algorithm 1 to compute c_{rb} from c_r and b
7: $c_{rs} \leftarrow c_{rb}$
8: **else if** $b = 0$ **then**
9: Use Algorithm 1 to compute c_{ra} from c_r and a
10: $c_{rs} \leftarrow c_{ra}^T$
11: **else**
12: Compute $s_u = [c_{r(T-2)}, c_{r(T-1)}, c_{rT}, c_{r(T+1)}]$ from c_r and T
13: $s_{k,l} \leftarrow s_u$, $c_l \leftarrow c_r$
14: Use Algorithm 2 to compute c_{ak+bl} from a, b, c_l and $s_{k,l}$
15: $c_{rs} \leftarrow c_{ak+bl}$
16: **end if**
17: **end if**
18: Return c_{rs}

the two input exponents is approximately an $\lfloor i \rfloor$-bit integer. If $s \approx n$ and if one uses Algorithm 1 to compute $[c_{r(T-2)}, c_{r(T-1)}, c_{rT}, c_{r(T+1)}]$ in step 12 of Algorithm 3 then the running time of Algorithm 3 is approximately $C_1((\log_2 n)/2) + C_2((\log_2 n)/2)$ since $T \approx \sqrt{n}$. Therefore, we can conclude from Table 3.1, and from the running time analysis of Algorithm 2 at the end of Section 4, that the running time of Algorithm 3 will not exceed $(4\log_2 n + 12.4\log_2 n)M_1/2 = 8.2(\log_2 n)M_1$ and the running time of Algorithm 3 is expected to be $(4\log_2 n + 6.37\log_2 n)M_1/2 \approx 5.19(\log_2 n)M_1$. Moreover, in the case that the base c_r is fixed, $s_{k,l}$ in Algorithm 3 can be precomputed, and the running time of Algorithm 3 is expected to be $6.38(\log_2 n)M_1/2 = 3.19(\log_2 n)M_1$. Assuming the base c_r is fixed, we may conclude that Algorithm 3 is faster than Algorithm 1. However, for general bases, Algorithm 1 remains the fastest single exponentiation algorithm in factor-4 groups, unless we can find a more efficient method for computing $[c_{r(T-2)}, c_{r(T-1)}, c_{rT}, c_{r(T+1)}]$ given c_r and T.

We now argue that $[c_{r(T-2)}, c_{r(T-1)}, c_{rT}, c_{r(T+1)}]$ can be computed at a negligible cost. This will refine Algorithm 3 and give a faster single-exponentiation algorithm than Algorithm 1 for general bases. We first need the following theorem.

Theorem 5.1. *Let $\mu_n = \langle g \rangle \subset \mathbb{F}_{q^4}^*$ be a factor-4 group with trace t, and let $c_r = \mathrm{Tr}(g^r)$. Then*
(i) $c_{rt} = c_r^t$.
(ii) $c_{r(t-1)} = c_r$.
(iii) $c_{r(t-2)} = c_r^t$.
(iv) $c_{r(t+1)}$ is a root of $F(x) = x^2 + c_r^{t+1}x + (c_r^{t+2} + c_r^{3t} + c_r^2 + c_r^4)$.

Proof. (i) follows because t is a power of 2 and $\mathrm{char}(\mathbb{F}_q) = 2$.
(ii) $c_{r(t-1)} = c_{rq} = c_r^q = c_r$ since $q \equiv t - 1 \pmod{n}$.

(iii) $c_{r(t-2)} = c_{rqt} = c_r^t$ since $qt \equiv q^2 + q \equiv q - 1 \equiv t - 2 \pmod{n}$.
(iv) First note that

$$c_{r(2t-1)} = \frac{1}{c_r^{t+1}}\left((c_{r(t+1)} + c_{rt} + c_{r(t-1)} + c_{r(t-2)})^2\right) + \frac{1}{c_r^{t+1}}\left((c_{rt} + c_{r(t-1)})^2(c_r^t + c_{rt}^2)\right), \tag{5.1}$$

by using Theorem 3.4(iii) with $u = rt$ and $v = r$. Moreover, using Lemma 3.2(iv) with $u = rt$ and $v = r$ together with the fact that $c_{r(t-1)} = c_r$ from part (ii), we can show that

$$c_{r(2t-1)} = c_r^{t+1} + c_{r(t+1)}. \tag{5.2}$$

Now, combining (5.1) and (5.2) we can write

$$c_{r(t+1)}^2 + c_r^{t+1} c_{r(t+1)} + (c_r^{t+2} + c_r^{3t} + c_r^2 + c_r^4) = 0,$$

which proves the result. □

Suppose now that c_r is known and that $t > 0$, whence $T = t$ (we can similarly argue when $t < 0$). By Theorem 5.1(i), (ii), and (iii), we can compute $[c_{r(T-2)}, c_{r(T-1)}, c_{rT}]$ at a cost of $2F_1$. We also know that the roots of $F(x) = x^2 + Bx + C$, $B, C \in \mathbb{F}_q$, $B \neq 0$, are given by $\{r_1, r_2\} = \{B \cdot R(C/B^2), B \cdot (R(C/B^2) + 1)\}$, where $R(C/B^2)$ is a root of $x^2 + x + (C/B^2)$, and if $r_1, r_2 \in \mathbb{F}_q$ then they can be computed at a negligible cost (see [10, Section 3.6.2]). By Theorem 5.1(iv), the roots of $F(x) = x^2 + c_r^{t+1}x + (c_r^{t+2} + c_r^{3t} + c_r^2 + c_r^4)$ are in $\mathbb{F}_q$, and so can be computed efficiently. Hence, we can determine $[c_{r(T-2)}, c_{r(T-1)}, c_{rT}, c_{r(T+1)}]$ at a negligible cost, up to the ambiguity that we cannot differentiate between the roots $c_{r(T+1)}$ and $c_{r(T+1)} + c_r^{T+1}$ of $F(x)$ (see Theorem 5.1). This ambiguity problem can be resolved if the sender of c_r is also required to compute $c_{r(T+1)}$ and send one extra bit $b \in \{0, 1\}$ to help the receiver distinguish $c_{r(T+1)}$ from $c_{r(T+1)} + c_r^{T+1}$ (see Section 5.2 for a similar discussion on how b can be determined).

Hence, assuming that we have a general base c_r and that the distinguisher bit b is known, the running time of Algorithm 3 does not exceed $(12.4 \log_2 n)M_1/2 = 6.2(\log_2 n)M_1$. Based on our experiments (see Table 4.3), the running time of Algorithm 3 is expected to be $(6.37 \log_2 n)M_1/2 \approx 3.19(\log_2 n)M_1$; this is 20% faster than Algorithm 1 which requires a negligible precomputation and has running time $4(\log_2 n)M_1$ (see Table 3.1). Note that the Diffie-Hellman key exchange protocol is an example of the scenario where c_r and b can be computed from c_1 by one of the communicating parties and sent to the other party.

To be more concrete, we compare the expected running time of our new single-exponentiation algorithm (Algorithm 3) with the previously known fastest single-exponentiation algorithm (Algorithm 1) at the 128-bit security level when general bases are employed. We let $q = 2^{1223}$ and $t = 2^{612}$. Then $q + 1 - t = 5n$ where n is a 1221-bit prime, and $\mathbb{F}_{q^4}^*$ has a factor-4 subgroup μ_n of order n. For any exponent $s \in [1, n-1]$, we can write $s = a \cdot 2^{612} + b$ where a has bitlength at most 609 and b has bitlength at most 612, and compute c_{rs} using Algorithm 3 with input c_r, s and these a, b values. Based on our experiments, the average cost of Algorithm 3 is $3895M_1$ (see Table 4.3). Note that this is 20% less than the cost of Algorithm 1, which is $4880M_1$.

5.2. **Nyberg-Rueppel Signature Scheme in Factor-4 Groups.** The Nyberg-Rueppel signature scheme [16] can be modified to be used with compressed representations of elements in μ_n as follows. We suppose that q, n and c_1 are system-wide parameters; alternately, they may be included in a party's public key. Alice chooses a random integer $k \in [1, n-1]$ and computes $s_{k,1} = [c_{k-2}, c_{k-1}, c_k, c_{k+1}]$. Alice's public key is $s_{k,1}$ and her private key is k. To sign a message M, Alice chooses a random integer $a \in [1, n-1]$ and computes c_a. Alice extracts a session key $K = \mathrm{Ext}(c_a)$ from c_a, and uses a symmetric-key encryption function E to encipher M under K: $e = \mathrm{E}_K(M)$. Moreover, she computes the hash $h = \mathrm{H}(e)$ of the encrypted text and sets $s = k \cdot h + a \bmod n$. Alice's signature on M is (e, s).

If Bob wants to verify Alice's signature (e, s) on M, he first computes $h = \mathrm{H}(e)$, replaces h by $-h \bmod n$, and computes c_{hk+s} from $s_{k,1}$ and c_1. Note that this is a double-exponentiation to the base $(k, 1)$. Bob extracts the session key $K' = \mathrm{Ext}(c_{hk+s})$ from c_{hk+s} and computes $e' = \mathrm{E}_{K'}(M)$. Bob accepts the signature if and only if $e' = e$.

One advantage of using the compressed representation of elements in μ_n with the Nyberg-Rueppel signature scheme is that c_1 is a system parameter instead of (the longer) g. We further show that it is possible to have a shorter public key at the expense of some negligible precomputation. In particular, we show that c_{k+1} is a root of a certain quadratic polynomial P_k whose coefficients are determined by c_1, c_{k-1} and c_k. That is, Alice can omit c_{k+1} from her public key and instead specify one bit to help Bob distinguish c_{k+1} from the other root of P_k.

Consider the matrix

$$M_u = \begin{pmatrix} c_u & c_{u+1} & c_u & c_{u+1} \\ c_{u+1} & c_{u+2} & c_{u+3} & c_{u+4} \\ c_{u+2} & c_{u+3} & c_{u+4} & c_{u+5} \\ c_{u+3} & c_{u+4} & c_{u+5} & c_{u+6} \end{pmatrix}.$$

Giuliani and Gong [7] showed that the characteristic polynomial of the matrix $M_u^{-1} \cdot M_{u+k}$ is equal to the characteristic polynomial of g^k over $\mathbb{F}_q$, namely $f_{g^k}(x) = x^4 + c_k x^3 + c_k^t x^2 + 1$. In particular, using Lemma 3.2(i), we can compute the characteristic polynomial P_k of $M_{-2}^{-1} \cdot M_{k-2}$ and find that the coefficient $C_2(P_k)$ of x^2 in $P_k(x)$ satisfies

$$\begin{aligned} (c_1 c_t)^2 C_2(P_k) &= (c_1^2 + c_t^2)c_{k+1}^2 + (c_1 c_t((c_{k-2} + c_k) + c_1 c_{k-1} + c_k c_t))c_{k+1} \\ &\quad + (c_1(c_{k-2} + c_k))^2 + c_1^2 c_t(c_{k-2} c_k + (c_{k-1} + c_k)^2) \\ &\quad + c_1 c_{k-1} c_t(c_1^2 c_k + c_{k-2} + c_k) + (c_{k-1}(c_1 + c_1^2 + c_t))^2 \\ &\quad + c_k^2(c_1^4 + c_t^3). \end{aligned}$$

We should note that $c_1 \neq 0$ as otherwise f_g would not be irreducible over $\mathbb{F}_q$. Since $P_k = f_{g^k}$, we must have $C_2(P_k) = c_k^t$ yielding the following result.

Theorem 5.2. *c_{k+1} is a root of the polynomial $P_k(x) = Ax^2 + Bx + C$ where*

$$\begin{aligned} A &= c_1^2 + c_t^2 \\ B &= c_1 c_t((c_{k-2} + c_k) + c_1 c_{k-1} + c_k c_t) \\ C &= (c_1(c_{k-2} + c_k))^2 + c_1^2 c_t(c_{k-2} c_k + (c_{k-1} + c_k)^2 + c_k^t c_t) \\ &\quad + c_1 c_{k-1} c_t(c_1^2 c_k + c_{k-2} + c_k) + (c_{k-1}(c_1 + c_1^2 + c_t))^2 + c_k^2(c_1^4 + c_t^3). \end{aligned}$$

First note that $A = 0$ if and only if $c_1 = c_t$, that is, if and only if g and g^t are conjugates over $\mathbb{F}_q$. Using $q^2 \equiv -1 \pmod{n}$ and $q \equiv t - 1 \pmod{n}$, we can show that this is never the case.

Now, the roots of $P_k(x) = Ax^2 + Bx + C$, $B, C \in \mathbb{F}_q$, $A, B \neq 0$, are given by $\{r_1, r_2\} = \{(B/A)\cdot R(AC/B^2), (B/A)\cdot(R(AC/B^2)+1)\}$, where $R(AC/B^2)$ is a root of $x^2+x+(AC/B^2)$. Furthermore, if $r_1, r_2 \in \mathbb{F}_q$ then they can be computed at a negligible cost (see [10, Section 3.6.2]).

If $A, B \neq 0$, then Alice's public key can be $([c_{k-2}, c_{k-1}, c_k], b)$ where $b \in \{0, 1\}$ is determined by Alice to help Bob to distinguish c_{k+1} from the other root of P_k. For example, Alice can do this as follows. She first computes c_{k+1} (c_{k+1} is obtained for free while computing c_k from c_1 using a single-exponentiation algorithm) and determines the roots r_1, r_2 of P_k together with integers corresponding to the bit representations of r_1 and r_2, say i_1 and i_2, respectively. We may assume without loss of generality that $i_1 < i_2$. If $c_{k+1} = r_1$ then Alice sets $b = 0$, otherwise she sets $b = 1$. Given Alice's public key, Bob can easily recover $s_{k,1}$ at a negligible cost and verify Alice's signatures.

If $A \neq 0$ and $B = 0$ in $P_k(x)$, then c_{k+1} can be uniquely computed by taking the square root of C/A in $\mathbb{F}_q$ at a negligible cost.

To be more concrete, we compare the length of public keys when compression is deployed with the length of public keys in the original scheme at the 128-bit security level. As in Section 5.1, we let $q = 2^{1223}$ and $t = 2^{612}$, whence $q+1-t = 5n$ where n is a 1221-bit prime and $\mathbb{F}_{q^4}^*$ has a factor-4 subgroup μ_n of order n. We find that the length of a public key when compression is deployed is 3669 bits, while the public key length without compression is 4892 bits.

6. Concluding Remarks

We showed that double-exponentiation can be efficiently performed using factor-4 compressed representation of elements in factor-4 groups. This allowed us to speed up single-exponentiation and also to use compression techniques in ElGamal type signature schemes.

A future research goal is to further speed up single-exponentiation and double-exponentiation algorithms that use compressed representation of elements. One possibility is that Algorithm 2 can be optimized by taking into account update rules different from those given in Table 4.1 (see for example [15] and [22]). Another possibility is to search for better parameters. For example, at the 128-bit security level, generating parameters q, n such that $\mu_n \subset \mathbb{F}_{q^4}^*$ is a factor-4 group, $q = 2^m \approx 2^{1200}$, and n is a 256-bit prime, would result in shorter exponents and therefore a speed up by a factor of more than 4. Such parameters can be found by searching for a suitable prime factor n of $N_1 = \gcd(\Phi_{4m}(2), q + 1 - t)$ or $N_2 = \gcd(\Phi_{4m}(2), q + 1 + t)$, where Φ_{4m} is the $(4m)$th-cyclotomic polynomial of degree $\varphi(4m)$, and φ is Euler's totient function. Note that when $\varphi(m)$ is significantly smaller than m, then factoring N_i is expected to be easier than factoring $q + 1 \pm t$. For example, when $m = 1209$, N_2 is a 718-bit integer and has a 271-bit prime factor n.

We expect that our double-exponentiation algorithm can be adapted to the factor-6 groups that arise as multiplicative subgroups of characteristic three finite fields [20, 11]. However, further analysis would be needed to estimate its efficiency, and to judge how the resulting single-exponentiation method compares to previously known algorithms.

Acknowledgments

The author thanks Alfred Menezes for reading earlier drafts of this paper and for his valuable remarks and suggestions.

References

[1] A. Brouwer, R. Pellikaan, and E. Verheul, *Doing more with fewer bits*, Advances in Cryptology – ASIACRYPT '99, Lecture Notes in Computer Science **1716** (1999), 321–332.

[2] D. Coppersmith, *Fast evaluation of logarithms in fields of characteristic two*, IEEE Transactions on Information Theory **30** (1984), 587–594.

[3] W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), 644–65.

[4] M. Dijk, R. Granger, D. Page, K. Rubin, A. Silverberg, M. Stam, and D. Woodruff, *Practical cryptography in high dimensional tori*, Advances in Cryptology – EUROCRYPT 2005, Lecture Notes in Computer Science **3494** (2005), 234–250.

[5] M. Van Dijk and D. Woodruff, *Asymptotically optimal communication for torus-based cryptography*, Advances in Cryptology – CRYPTO'2004, Lecture Notes in Computer Science **3152** (2004), 151–178.

[6] FIPS 186-3, *Digital signature standard (DSS)*, Federal Information Processing Standards Publication 186-3, National Institute of Standards and Technology, 2009.

[7] K. Giuliani and G. Gong, *New LFSR-based cryptosystems and the trace discrete log problem (Trace-DLP)*, Sequences and Their Applications – SETA 2004, Lecture Notes In Computer Science **3486** (2004), 298–312.

[8] G. Gong and L. Harn, *Public-key cryptosystems based on cubic finite field extensions*, IEEE Transactions on Information Theory **45** (1999), 2601–2605.

[9] D. Gordon, *Discrete logarithms in $GF(p)$ using the number field sieve*, SIAM Journal on Discrete Mathematics **6** (1993), 124–138.

[10] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*, Springer-Verlag, New York, NY, USA, 2004.

[11] K. Karabina, *Factor-4 and 6 compression of cyclotomic subgroups of $\mathbb{F}^*_{2^{4m}}$ and $\mathbb{F}^*_{3^{6m}}$*, Journal of Mathematical Cryptology, to appear. Earlier version available at `http://eprint.iacr.org/2009/304`.

[12] A. Lenstra, *Unbelievable security matching AES security using public key systems*, Advances in Cryptology – ASIACRYPT 2001, Lecture Notes in Computer Science **2248** (2001), 67–86.

[13] A. Lenstra and E. Verheul, *The XTR public key system*, Advances in Cryptology – CRYPTO 2000, Lecture Notes in Computer Science **1880** (2000), 1–19.

[14] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of applied cryptography*, CRC Press, New York, NY, USA, 1997.

[15] P. Montgomery, *Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains*, `www.cwi.nl/ftp/pmontgom/Lucas.ps.gz`, December 13, 1983; Revised March, 1991 and January, 1992.

[16] K. Nyberg and A. Rueppel, *Message recovery for signature schemes based on the discrete logarithm problem*, Designs, Codes and Cryptography **7** (1996), 61–81.

[17] J. Pollard, *Monte Carlo methods for index computation mod p*, Mathematics of Computation **32** (1978), 918–924.

[18] K. Rubin and A. Silverberg, *Compression in finite fields and torus-based cryptography*, SIAM Journal on Computing **37** (2008), 1401–1428.

[19] M. Scott, *Authenticated ID-based key exchange and remote log-in with simple token and PIN number*, Cryptology ePrint Archive, Report 2002/164, 2002, `http://eprint.iacr.org/2002/164`.

[20] M. Shirase, D. Han, Y. Hibin, H. Kim, and T. Takagi, *A more compact representation of XTR cryptosystem*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **E91-A** (2008), 2843–2850.

[21] P. Smith and C. Skinner, *A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms*, Advances in Cryptology – ASIACRYPT '94, Lecture Notes In Computer Science **917** (1994), 357–364.

[22] M. Stam and A. Lenstra, *Speeding up XTR*, Advances in Cryptology – ASIACRYPT 2001, Lecture Notes in Computer Science **2248** (2001), 125–143.

Dept. of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

E-mail address: kkarabin@uwaterloo.ca